REMARKS

This Amendment is in response to the Final Office Action dated July 5, 2005. In the Office Action, the Examiner rejected claims 1-24 under 35 U.S.C. § 103(a) as being unpatentable over Fumer *et al.*, U.S. Patent No. 5,974,474 (hereinafter *Fumer*), in view of Dinallo, U.S. Patent No. 5,727,212 (hereinafter *Dinallo*). Claims 5 and 6 were rejected under 35 U.S.C. § 112 for improper antecedent basis.

Claim 5 is amended, as shown above, to correct the antecedent basis problem. No other amendments are made herein, and no claims are cancelled or added. Accordingly, claims 1-24 remain pending in the application. For the reasons set forth below, the Applicants respectfully request reconsideration and allowance of all pending claims.

Traversal of the Claim Rejections under 35 U.S.C. § 103

To establish a *prima facie* case of obviousness, there must first be some suggestion or motivation to modify a reference or to combine references, and second be a reasonable expectation of success. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art and not based on applicant's disclosure. Third, the prior art reference (or references when combined) must teach or suggest all the claim limitations. M.P.E.P. § 706.02(j) from *In Re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991). Where claimed subject matter has been rejected as obvious in view of a combination of prior art references, a proper analysis under § 103 requires, *inter alia*, consideration of two factors: (1) whether the prior art would have suggested to those of ordinary skill in the art that they should make the claimed device; and (2) whether the prior art would also have revealed that in so making, those of ordinary skill would have a reasonable expectation of success. Both the suggestion and the reasonable expectation of success must be founded in the prior art, not in the Applicants' disclosure. *Amgen v. Chugai Pharmaceutical*, 927 F.2d 1200, 18 USPQ2d 1016 (Fed. Cir. 1991), *Fritsch v.*

Lin, 21 USPQ2d 1731 (Bd. Pat. App. & Int'f 1991). An invention is non-obvious if the references fail not only to expressly disclose the claimed invention as a whole, but also to suggest to one of ordinary skill in the art modifications needed to meet all the claim limitations. *Litton Industrial Products, Inc. v. Solid State Systems Corp.*, 755 F.2d 158, 164, 225 USPQ 34, 38 (Fed. Cir. 1985).

The examiner must present a convincing line of reasoning as to why the artisan would have found the claimed invention to have been obvious in light of the teachings of the references. M.P.E.P. § 70602(j) from *Ex parte Clapp*, 227 USPQ 972, 973 (Bd. Pat. App. & Inter. 1985). Obviousness cannot be established by combining references without also providing evidence of the motivating force which would impel one skilled in the art to do what the patent applicant has done. M.P.E.P. § 2144 from *Ex parte Levengood*, 28 USPQ2d 1300, 1302 (Bd. Pat. App. & Inter. 1993) (emphasis added by M.P.E.P.).

Claims 1-24 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over *Furner* in view of *Dinallo*.

Claim 1 is illustrative of the claimed invention, and recites,

1. A method for representing a root bus of a computer system, comprising:

dynamically generating an object-oriented abstraction corresponding to the root bus *referencing* one or more *methods* that may be implemented to obtain and/or generate configuration and resource allocation information *for the root bus and any subordinate busses connected either directly or indirectly to the root bus*; and

registering the methods referenced in the object-oriented abstraction via a data structure stored in memory of the computer system. (Emphasis Added)

In support of the § 103(a) rejection of claim 1, the Examiner states,

> Regarding claim 1, Furner et al. disclose a method for representing a root bus, comprising:
>
> dynamically [generating an object-oriented abstraction corresponding to the root bus referencing one or more methods that may be implemented] to obtain and/or generate configuration and resource

allocation information for the root bus and any subordinate busses connected either directly or indirectly to the root bus (configuration process includes resolving conflicts, FIG. 13); and

Furner et al. also disclose peripheral buses are characterized by being connected, directly or indirectly, to the CPU-memory bus through bus controllers that actively manage the communication to the hardware devices on the bus (column 10, lines 13-16). Furthermore, Furner et al. disclose an installation information table as shown in FIG. 2E, which implies registering functions. However, Furner et al. fail to expressly disclose defining an object-oriented abstraction including methods. Nevertheless, Furner et al. do suggest the abstracting into functional modules. Thus, applications could refer to hardware instances in a common manner (column 31, lines 41-51).

Dinallo discloses bridging communication between an object oriented component and a procedural programmed device driver (Dinallo, column 2, lines 4-16). As shown in Fig. 3, Object includes multiple methods to provide different functions. In the OOP environment, registering the methods is well known.

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the teachings of Furner et al. to incorporate the teachings of Dinallo to obtain the invention as specified in claim 1 because bridging communication between an object oriented component and a procedural programmed device driver the existing procedural programmed device driver could be reused in the OOP environment.

Furner uses an identification and configuration system 119 to identify hardware devices and hardware instances of those devices. With respect to the Examiner's comment of, "Furner et al. disclose an installation information table as shown in FIG. 2E, which implies registering functions," the installation information table stores information related to hardware device drivers. In further detail, *Furner* states,

> As will be explained in detail below, the identification and configuration system 119 of the present invention determines the optimal driver 121 for a particular hardware instance 150 by comparing various characteristics of all the drivers that are capable of supporting the hardware instance. To determine which drivers 121 can support a particular hardware instance, the identification and configuration system 119 compares the information in each driver record 241 with that in the hardware device records 240. *This process identifies drivers 121 that can support the particular hardware instance 150.* (Col 15, lines 8-18, emphasis added)

and

> Once an optimal *driver* 121 is selected, the identification and configuration system 119 places installation information for the selected *driver* into an *installation information table* 133 shown in FIG. 2E. The installation table 133 includes such information as the driver name 207, driver location 208, resource settings 142, and HIN number 222. *The identification and configuration system 119 uses the installation information table 133 to configure the hardware instance 150 in a manner described below.* (Col 15, lines 46-54, emphasis added)

It is clear from above that the installation information table 133 stores information identifying a selected driver for a corresponding hardware instance (*e.g.*, a hardware device driver). Such a driver is employed to access and configure its corresponding hardware instance. *Fumer* does not store any dynamically generated data relating to "one or more *methods* that may be implemented to obtain and/or generate configuration and resource allocation information *for the root bus and any subordinate busses connected either directly or indirectly to the root bus*. Under *Fumer*, configuration and resource allocation information for root busses and subordinate busses is obtained by the identification and configuration system 119, which includes predefined processes and methods to read data from and write data to hardware registers to identify resource requirements and usage. There is no dynamic identification of a method or methods to perform this operation, or registration of such methods in memory – the process and method are known *a priori* (in advance). The only functions that are registered under *Fumer* relate to drivers specific to respective hardware devices and/or hardware instances.

The Examiner further makes reference to Col. 31, lines 40-51, which states,

> There are portions of this identification and configuration system 119 which could be abstracted into functional modules, as in a functional programming language or object-oriented programming language such as C++, such as the automatic identification mechanism 301 and the configuration mechanism 302. These functional modules could be made available to applications such as installation utilities as shown above, monitoring software and network management software *for the purpose of identifying and configuring hardware instances.* Thus, applications could refer to hardware instances in a common manner. (Emphasis added)

The foregoing explicitly states that the functional modules could be employed *for the purpose of identifying and configuring hardware instances*. The one or more methods recited in claim 1 concern methods that are use to obtain and/or generate configuration and resource allocation information *for the root bus and any subordinate busses connected either directly or indirectly to the root bus*. Clearly, the methods referred to in claim 1 serve an entirely different function than the device drivers employed by *Furner* (which conceivably could be accessed via the theoretical functional modules discussed above).

With respect to this last argument, which was also made in the Response mailed by Applicants on March 25, 2005, the Examiner states in paragraph 9-2,

> Response to Applicants' arguments (2) and (3). By way of identifying and configuring for hardware instances, configuration and resource allocation information are generated at each associated table. For example, a number of resource settings 142 allocate system resources to a hardware instance 150 (column 12, lines 19-51). In other words, the disclosure of Furner et al. is not drivers only as asserted by the Applicants.

Whether or not *Furner* discloses configuration information for drivers only is not the issue here. The issue from above that was not addressed by the Examiner's response is whether *Furner* discloses dynamic generation of information (an objected-oriented abstraction in this particular case) identifying one or more methods that may be implemented to obtain and/or generate configuration and resource allocation information *for the root bus and any subordinate busses connected either directly or indirectly to the root bus.* Clearly Furner does not dynamically generate such information – the methods and processes used to obtain configuration and resource allocation information for the root bus and any subordinate busses are known in advance.

More specifically, *Furner* states the following with respect to obtaining bus configuration and hardware instance information:

As shown in FIG. 5, bus interface 503 provides access to hardware instances 150 for obtaining information from, and setting information in, the hardware devices 148. The bus interface 503 may be any *well-known bus interface* that accesses the installed hardware devices registers 125. In a preferred embodiment, the bus interface 503 is NetWare Bus Interface (NBI) available from Novell Incorporated, Provo, Utah. In an alternative embodiment, the bus interface 503 includes *well-known lower-level functions*. For example, when the hardware device 148 is an IDE bus controller, the hardware instance information is obtained by performing an IDE identify operation. When the hardware device 148 is a SCSI bus controller, information is obtained through a SCSI inquiry. *Hardware identification using these methods is well known in the art.* It should be understood that multiple techniques currently exist for obtaining hardware device information and thus, various techniques to obtain information may be used without varying from the scope of the present invention. (Col. 20, lines 28-47, emphasis added)

The process performed by the identification mechanism 501 will now be described with reference to FIG. 7. As discussed previously, the identification process 700 identifies all hardware instances in the computer system 100 and *optimally associates drivers to each of the hardware instances* 150. Process 700 begins at block 702 as part of the identification and configuration system 119. Once invoked, the identification process 700 advances to block 704 where at the bus interface 503 is loaded into run-time memory 101 to enable the present invention to obtain hardware instance information from the hardware devices 148. As discussed, *the bus interface 503 consists of processes which can read and write information to the registers 125 of the hardware devices 148.* (Col 20, line 60 to Col. 21, line 6, emphasis added)

At block 802, the primary buses are located and scanned for hardware instances 150 *using the bus interface 503*. In the embodiment wherein the bus interface 503 is NBI, the ScanBusInfo() and ScanCardInfo() methods are preferably used to iteratively search the primary buses for hardware instances 150 in a well-known manner. When provided with an initial sequence number value of -1, the ScanBusInfo() and ScanCardInfo() methods provide the first hardware instance 150 detected in the computer system 100. On subsequent searches, the sequence number from the previous search is provided as input for the next search performed by the ScanBusInfo() and ScanCardInfo() methods. The ScanCardInfo() method then retrieves the next hardware instance 150 in the computer system 100. The ScanCardInfo() method is used to obtain all of the hardware instances in the computer system 100 without having to discover the individual I/O buses that the ScanBusInfo() method discovers. Hence, a nested search loop for each I/O bus 105, 106 by the ScanBusInfo() method and hardware instance 150 by the ScanCardInfo() method is required. As noted, it should be understood that in other embodiments other types of bus interface 503 may be

> implemented. As one skilled in the relevant art would find apparent, in such embodiments a call would be made to the appropriate procedure(s) to obtain the above-noted hardware instance information. (Col. 22, lines 16-42, emphasis added)

Moreover, Fig. 5 (which is also shown on the cover sheet) shows the functional blocks of the *Fumer* identification and configuration system. As discussed above, bus interface 503 is used to read registers from and write information to various hardware instances using predefined functions (e.g., methods, procedures, etc.) that are configured and known in advance. None of the dynamically-generated information in the reference tables 117 pertains to the identification of methods "that may be implemented to obtain and/or generate configuration and resource allocation information for the root bus and any subordinate busses connected either directly or indirectly to the root bus."

More specifically, the reference tables 117 include hardware information table 129, driver information table 130, matched driver table 131, unmatched hardware table 132 and installation table 133. Details of hardware information table 129 are shown in Fig. 2A, driver information table 130 in Fig. 2B, matched driver table 131 in Fig. 2C, unmatched hardware table 132 in Fig. 2D and installation table 133 in Fig. 2E. It is clear from these figures and the accompanying text that none of these tables contains references to methods used to obtain and/or generate configuration and resource allocation information for the root bus and any subordinate busses. The only object in the tables akin to a method are hardware device drivers listed in match driver table 131, which as discussed above are employed for accessing hardware instances.

With respect to the Examiner's statement that *Dinallo* discloses bridging communication between an object-oriented component and a procedural programmed device driver, the Applicants agree with this argument in general. However, the one or more methods referred to in claim 1 do not relate to device drivers for hardware devices or instances, as discussed above. Clearly, since *Fumer* does not employ any

dynamically generated information identifying methods to obtain and/or generate configuration and resource allocation information for the root bus and any subordinate busses, there would be no motivation for anyone of ordinary skill in the art to employ an object-oriented abstraction for doing so in view of the teachings of *Dinallo*, nor any expectation of success.

In view of the foregoing argument it is clear that the combination of *Fumer* and *Dinallo* do not teach or suggest all of the elements and limitations recited in claim 1, as required by the third prong of the *In Re Vaeck* test. Accordingly, a rejection of claim 1, as amended, as being unpatentable over *Fumer* in view of *Dinallo* would be unsupported and thus improper. Accordingly, claim 1 is patentable over the cited art.

With respect to amended independent claim 19, this claim is a Beauregard claim reciting software (*i.e.*, computer-executable instructions) for performing operations analogous to the operations recited in amended claim 1. Accordingly, amended claim 19 is patentable over the combination of *Fumer* and *Dinallo*.

With respect to independent claim 9, this claim recites, in part,

defining an object oriented representation of each root bus comprising a set of components that includes references to *a plurality of methods that may be implemented to obtain and/or generate configuration and resource allocation information for that root bus and any subordinate busses connected either directly or indirectly to the root bus*;

assigning a bus identifier for each of the subordinate busses *through use of an enumeration process that implements one or more of the methods referenced by the object oriented representation of that root bus*; (Emphasis added)

Again, the methods referred to in independent claim 9 are not drivers for hardware devices/instances. Accordingly, it is clear that the combination of *Fumer* and *Dinallo* do not teach or suggest all of the elements and limitations recited in claim 9, as required by the third prong of the *In Re Vaeck* test. Thus, amended claim 9 is patentable over the cited art.

Conclusion

Overall, none of the references singly or in any motivated combination disclose, teach, or suggest what is recited in the independent claims. Thus, given the above remarks, independent claims 1, 9, and 19 are in condition for allowance. The dependent claims that depend directly or indirectly on these independent claims are likewise allowable based on at least the same reasons and based on the recitations contained in each dependent claim.

If the undersigned attorney has overlooked a teaching in any of the cited references that is relevant to the allowability of the claims, the Examiner is requested to specifically point out where such teaching may be found. Further, if there are any informalities or questions that can be addressed via telephone, the Examiner is encouraged to contact the undersigned attorney at (206) 292-8600.

Charge Deposit Account

Please charge our Deposit Account No. 02-2666 for any additional fee(s) that may be due in this matter, and please credit the same deposit account for any overpayment.

Respectfully submitted,

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN

Date: Sept. 1, 2005

R. Alan Burnett
Reg. No. 46,149

12400 Wilshire Boulevard
Seventh Floor
Los Angeles, CA 90025-1030